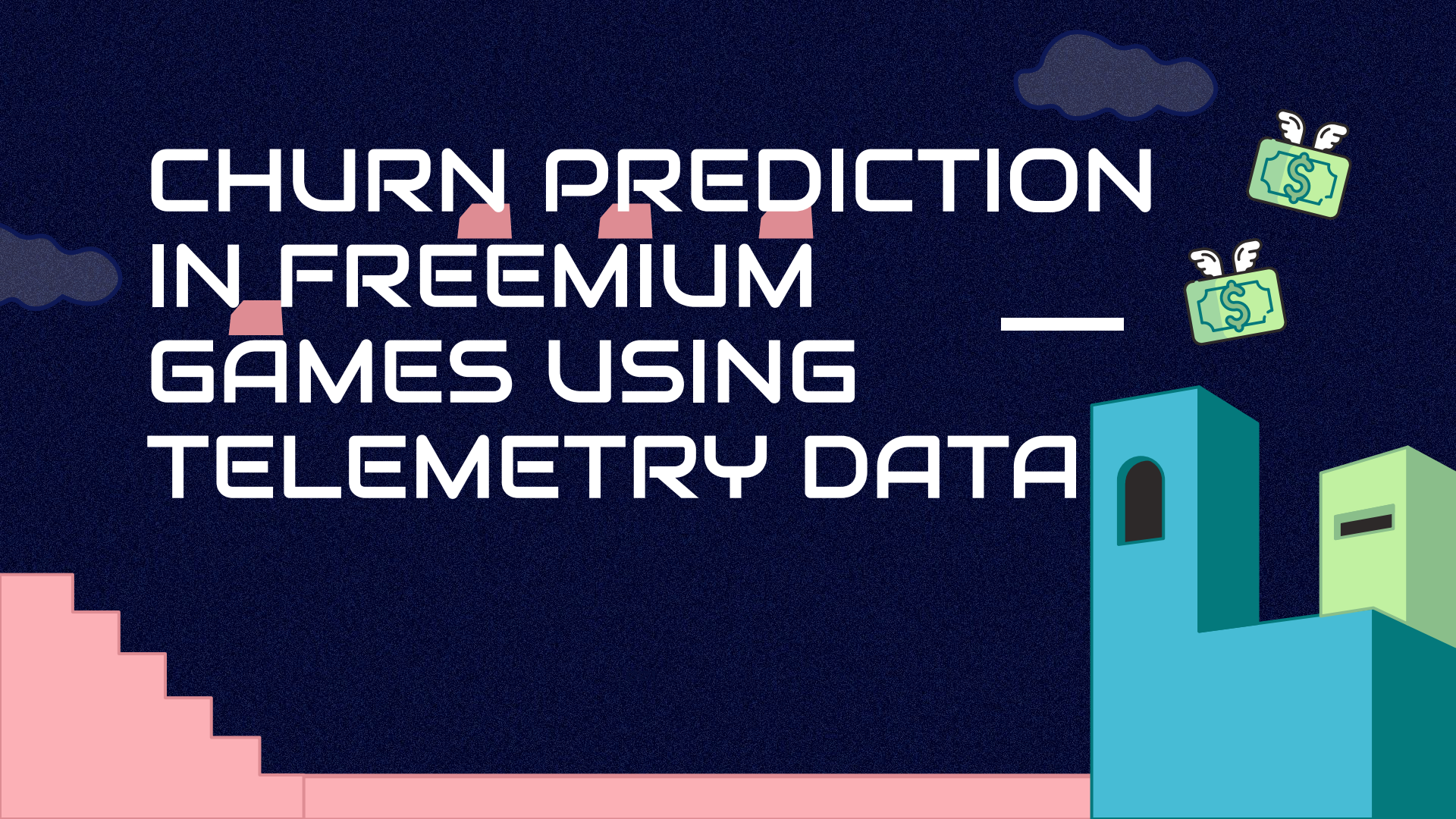
CHURN PREDICTION IN FREEMIUM GAMES USING TELEMETRY DATA



Churn: Inactivity of a player leading to quitting the game

GOAL: To predict whether a player will churn from a freemium mobile game based on their in-game behavior during an initial observation period

Assignment Tasks:

- Structuring the Dataset
- Building a Supervised Dataset (using the training set), Defining time periods & Labeling each player
- Feature Extraction
- Training 3 Classifiers
- Creating a new dataset (using the test set) & Performing Feature Extraction
- Model Evaluation (Best Classifier & Best Features)
- LLM evaluation and comparison

Task 1: Dataset Overview and Splitting:

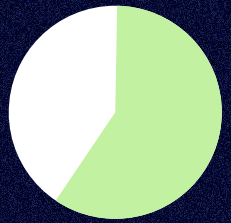
GAME 1 (.csv) → Json Object

{ Device ID: Player ID (unique)
Records: [List of {Score,Time} entries (dictionaries) for each ID] }

Game 2 (json) → Json Object

{ User ID: Player ID (Unique)
Records: [List of game event entries where each item in the list is a dictionary containing time, date, event type, properties {price, build, balance}, ...] }

The datasets are then split into:
80% ⟶ For Training
20% ⟶ For Testing/Evaluation



Using 80% (DS1)

Task 2: Building a labelled dataset using train_set

Defining the time windows

01

Task 2

Anchoring Player Timelines

02

Filtering & Labeling Churn

03

01 Defining the time windows:

Each Player's Activity is split into:

- Observation Period: Day 0 to Day 5
- Churning/Prediction Period: Following 10 days

02

Sorting & Anchoring Player Timelines

For Each player:

Sorting the events chronologically by timestamp

Identifying the 1st event and converting all absolute times to relative times

Eg.
abs_ts_scores = [(time1, score1), (time2, score2), ..., (timeN, scoreN)]

t1_abs = abs_ts_scores[0][0]
 # absolute timestamp of first play of a player

rel_ts_scores = [(t_abs - t1_abs, score) for t_abs, score in abs_ts_scores]
#converting each timestamp to relative time

03 Filtering and Labelling:

Filtering : Only keeping players with at least one play/event in the observation period

Labeling:

Churned = 0 : Player has been active in the prediction window

Churned = 1 : No activity in the prediction window

Task 3: Feature Extraction (DS1)

List of Features (Game 1):

- Play_count
- Active_duration_s
- Best_sub_mean_ratio
- Sd_score
- first_day_play_count
- play_frequency_per_day
- avg_time_between_plays
- Best_score_index
- ....

List of Features (Game 2):

- Play_count
- Active_duration_s
- Recency_last_purchase
- Purchase_rate
- consecutive_play_ratio
- first_day_play_count
- play_frequency_per_day
- total_soft_spent
- ....

Intuition behind feature selection

Game 1:

- Play_count = Higher numbers ⟶ more engagement
- Active_duration_s = Longer sessions ⟶ higher interest
- Sd_score ⟶ Low sd_score could mean difficulty
- First_day_play_count ⟶ Interest before release
- Play_frequency_per_day ⟶ Returns to game often
- Sd_gap ⟶ small gap between returns

Game 2:

- Active_duration_s ⟶ Longer sessions ⟶ higher interest
- Recency_last_purchase ⟶ made a recent purchase
- Purchase_rate ⟶ buys items quite often
- Consecutive_play_ratio ⟶ back-to-back sessions within a defined period
- Total_soft_spent ⟶ more investment

Feature Extraction (Code Snippet Game 1)

```
# Computing features & label (from OP_DAYS only)
play_count          = len(obs_recs)  # number of plays in the observation period
active_duration_s   = obs_recs[-1][0] if play_count > 1 else 0 #Duration between first and last play in the observation window.
churn               = 1 if len(pred_recs) == 0 else 0  # labelling 1 if no activity in prediction window, otherwise 0.
scores              = [s for t, s in obs_recs] #extracting the scores from the plays in the observation period
mean_score          = float(np.mean(scores))               # meanScore
sd_score            = float(np.std(scores, ddof=0))       # standard deviation of score
best_score          = float(max(scores))                   # bestScore
worst_score         = float(min(scores))                   # worstScore
best_idx            = scores.index(best_score)             # index (position) of the first occurrence of the best score
best_score_index    = best_idx / (play_count - 1) if play_count > 1 else 0 # Normalizing the best_idx to a value between 0 and 1 by dividing

# bestSubMeanCount = (bestScore - meanScore) / n
best_sub_mean_count = (best_score - mean_score) / play_count if play_count else 0 #How much better was the best score than average, per play
# bestSubMeanRatio = (bestScore - meanScore) / meanScore
best_sub_mean_ratio = (best_score - mean_score) / mean_score if mean_score else 0  #relative difference between best score and mean score, as

C    = 3600  # threshold in seconds for defining consecutive plays
gaps = [obs_recs[i+1][0] - obs_recs[i][0] for i in range(play_count - 1)] #list of time gaps (in seconds) between consecutive plays.
consec_count            = sum(1 for g in gaps if g <= C) #no. of time gaps <=3600s ie number of back-to-back plays that happened within one o
consecutive_play_ratio  = consec_count / (play_count - 1) if play_count > 1 else 0 # ratio (percentage) of gaps less than 1 hour (C)
median_gap              = float(np.median(gaps)) if gaps else 0  # median time between plays
gap_sd                  = np.std(gaps) if len(gaps)>0 else 0
```

Feature Extraction (Code Snippet Game 2)

```
#Scores
scores = [r for (_,_,_,_,r) in obs_plays if r is not None]  # non-null rewards
mean_score = float(np.mean(scores)) if scores else 0.0
sd_score   = float(np.std(scores, ddof=0)) if scores else 0.0
best_score = float(max(scores)) if scores else 0.0
worst_score= float(min(scores)) if scores else 0.0

best_idx = scores.index(best_score) if len(scores) > 1 else 0
best_score_index    = best_idx / (len(scores)-1) if len(scores) > 1 else 0.0
best_sub_mean_count = (best_score - mean_score) / len(scores) if scores else 0.0
best_sub_mean_ratio = (best_score - mean_score) / mean_score if mean_score else 0.0

#Gap between plays
gaps = [obs_plays[i+1][0] - obs_plays[i][0]
        for i in range(play_count - 1)]
consec_count           = sum(1 for g in gaps if g <= 3600)   # plays within 1 hour
consecutive_play_ratio = consec_count / (play_count - 1) if play_count > 1 else 0.0
median_gap             = float(np.median(gaps)) if gaps else 0.0

# Failure stats
failure_count = sum(1 for (_,_,action,fs,_) in obs_plays
                    if action in ("fail", "quit") and fs is not None)
failure_ratio = failure_count / play_count if play_count else 0.0

# Queue duration
qd_list = [qd for (_,qd,_,_,_) in obs_plays] + [qd for (t_rel,qd,_) in rel_buys if t_rel <= OP_S]
avg_queue_duration = float(np.mean(qd_list)) if qd_list else 0.0
```

Task 4: Model Training (Game 1)

Decision Tree 01

```python
base_tree = DecisionTreeClassifier(random_state=30)  #base decision tree

param_grid = {                                       #Defining hyperparameters to test
    'max_depth'        : [None, 5, 10, 20, 30], #max depth of tree
    'min_samples_split': [2, 10, 20, 30], #Minimum samples to split
    'min_samples_leaf' : [1, 5, 10, 15], #Minimum samples in a leaf
    'ccp_alpha'        : [0.0, 0.005, 0.01, 0.02] #Pruning parameters
}
```

Random Forest 02

XGBoost 03

Sklearn library

Task 4: Model Training (Game 2)

Decision Tree 01

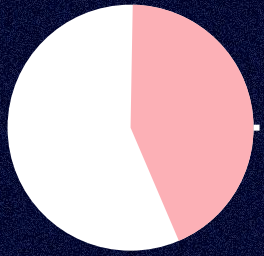
```
#
base_tree2 = DecisionTreeClassifier(random_state=30)  #base decision tree model
param_grid2 = {
    'max_depth'        : [None, 5, 10, 20],  #max depth of tree
    'min_samples_split': [2, 10, 20], #Minimum samples to split
    'min_samples_leaf' : [1, 5, 10],    #Minimum samples in a leaf
    'ccp_alpha'        : [0.0, 0.005, 0.01]    #Pruning parameter
}
```

Random Forest 02

Sklearn library

XGBoost 03

Task 5: Creating DS2 (Game 1 and 2)



Using 20%
(DS2)

Task 1 - Task 3
(Till Feature extraction)

Evaluation Metrics

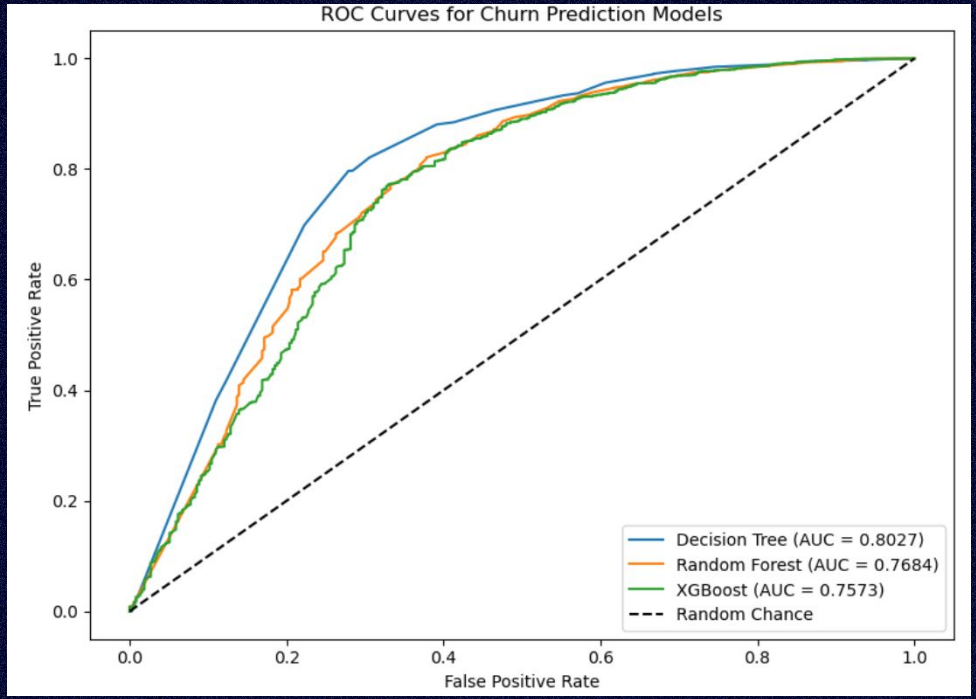
- Accuracy
- Precision
- Recall
- F1 Score
- Area Under ROC Curve

Task 6: Model Evaluation (Scores)

Game 1	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Decision Tree	0.930842	0.937930	0.991075	0.963770	0.802746
Random Forest	0.930071	0.939262	0.988584	0.963293	0.768450
XGBoost	0.930264	0.939795	0.988169	0.963375	0.757327

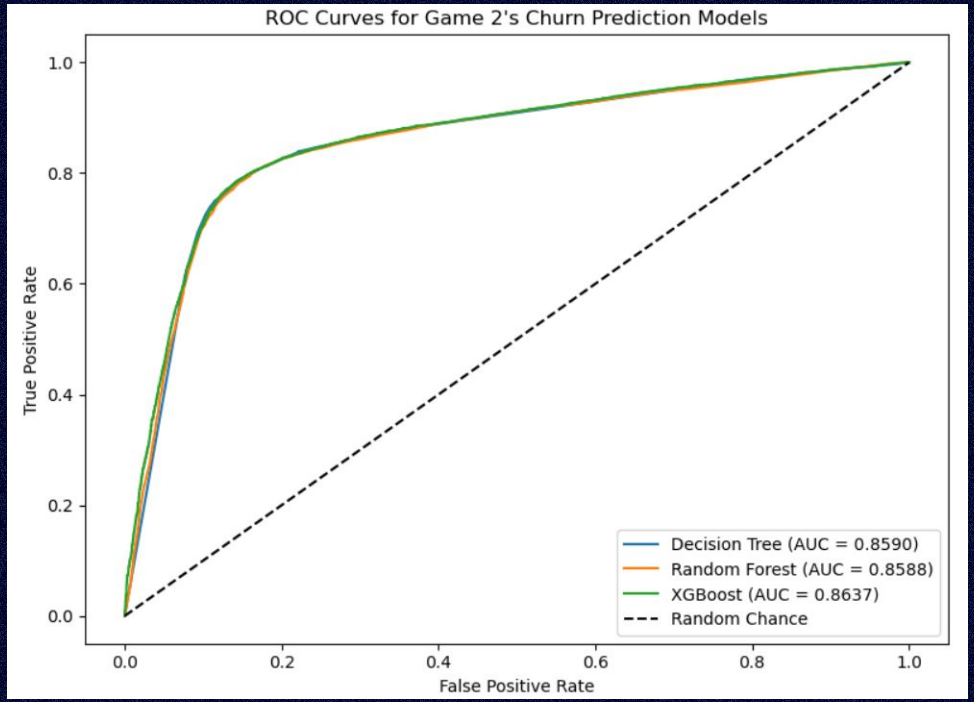
Game 2	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Decision Tree	0.815824	0.867206	0.812040	0.838717	0.858952
Random Forest	0.816265	0.870434	0.808838	0.838506	0.858836
XGBoost	0.815195	0.860877	0.818977	0.839405	0.863707

Task 6: Model Evaluation (ROC Curve & Best Performing Model - Game 1)



Best Performing Model: **Decision Tree (Based on AUC)**

Task 6: Model Evaluation (ROC Curve & Best Performing Model - Game 2)



Best Performing Model: **XGBoost (Based on AUC)**

Task 6: Model Evaluation (Single Feature Ranking)

1. Absolute Pearson Correlation with churn

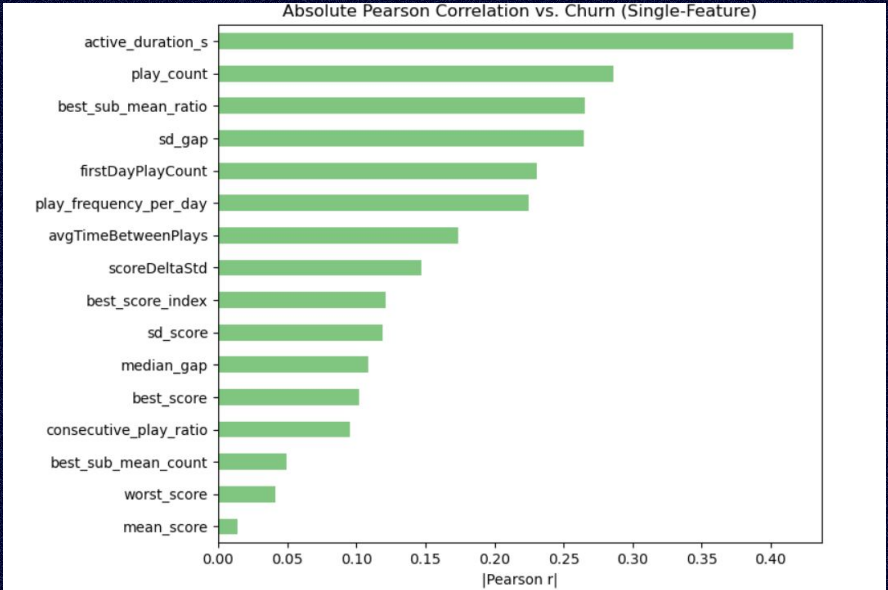
2. Single-feature AUC (Decision Tree) : 10-fold Cross validation AUC when the feature is used alone with the chosen Decision Tree model

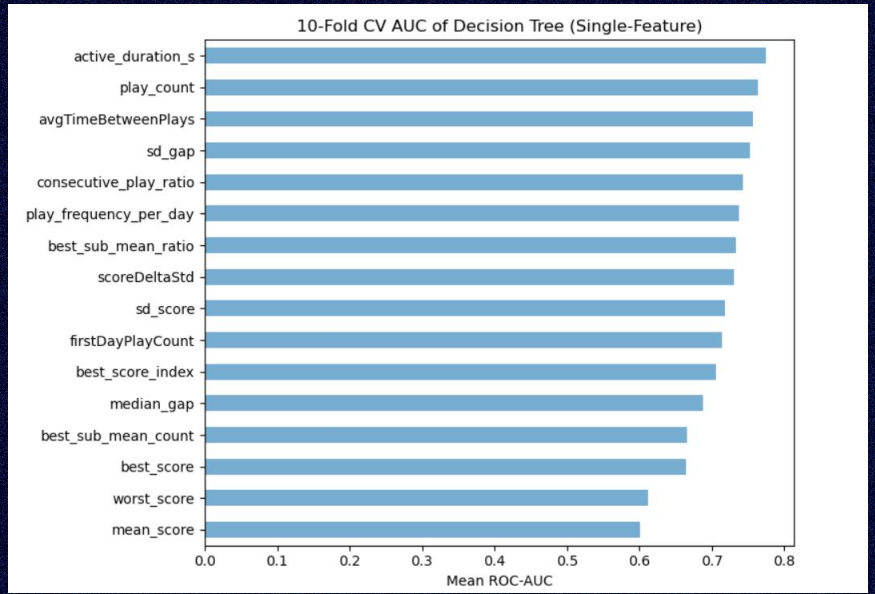
3. Single-feature AUC (Random Forest) : 10-fold Cross validation AUC when the feature is used alone with the chosen Random Forest model

4. Single-feature AUC (XGBoost) : 10-fold Cross validation AUC when the feature is used alone with the chosen XGBoost model

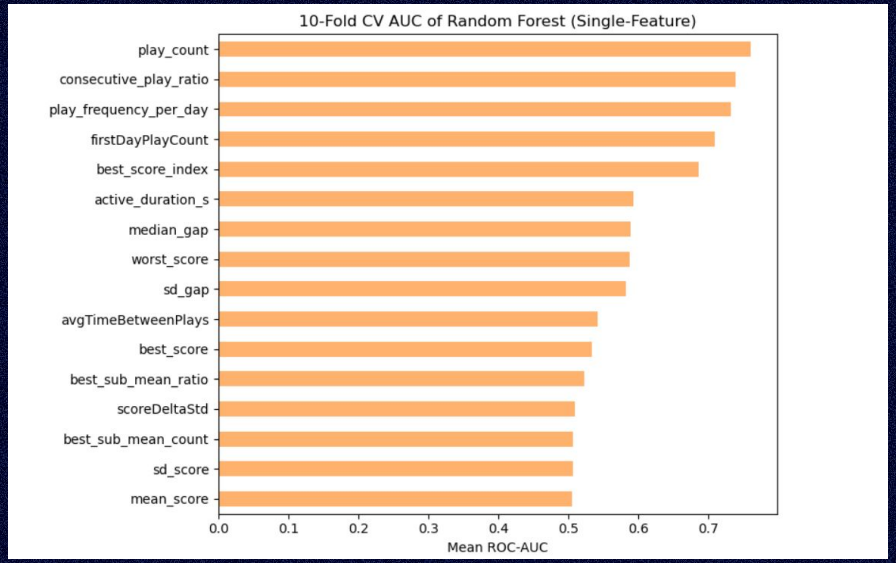
5. XGBoost Feature Importance : Individual importance score for each feature calculated by XGBoost after training on all features together

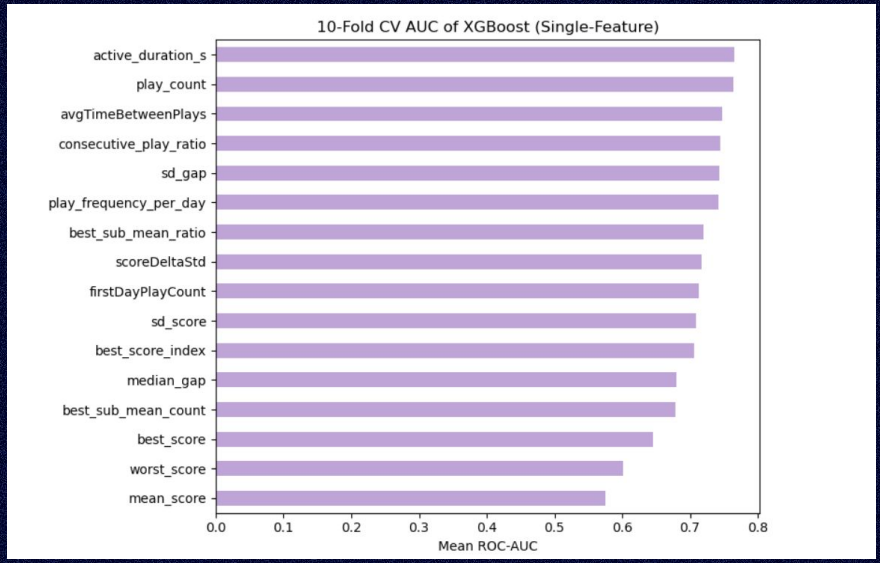
Task 6: Model Evaluation (Single Feature Ranking Charts - Game 1)



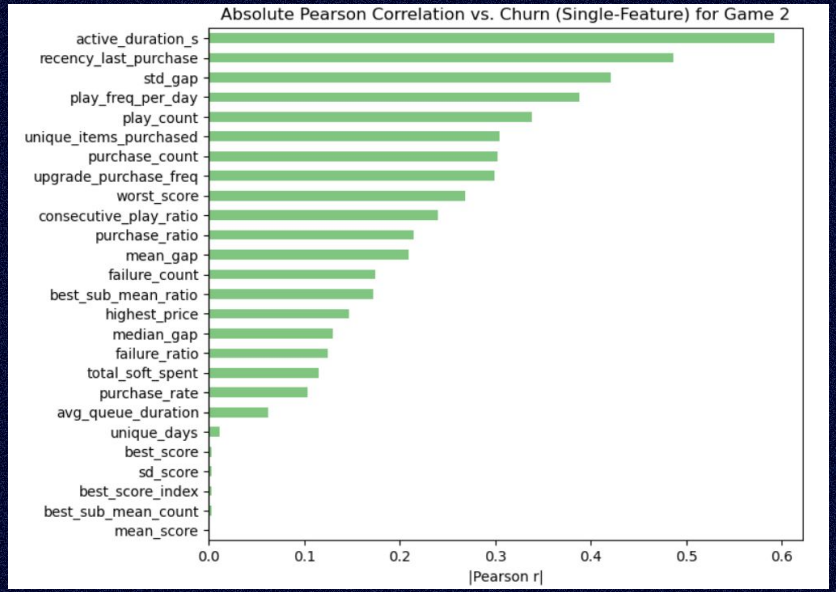


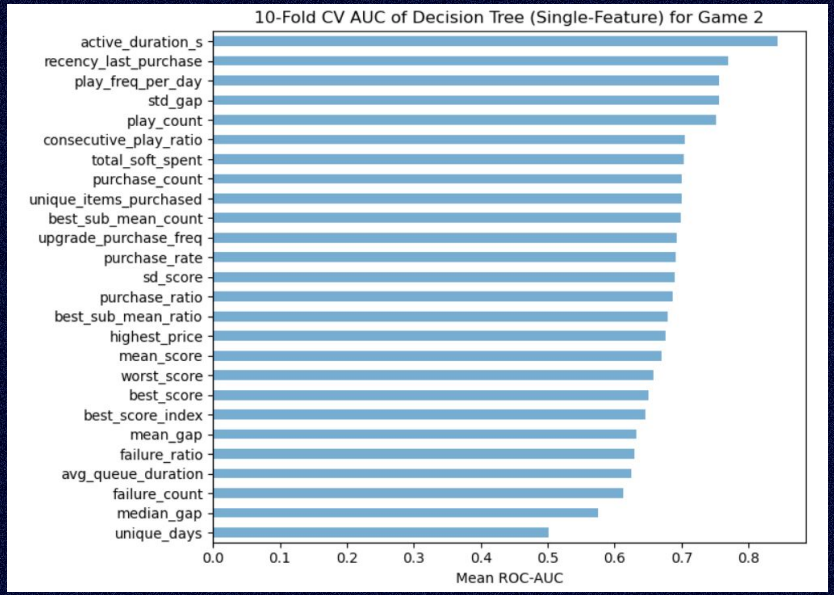
Task 6: Model Evaluation (Single Feature Ranking Charts - Game 1)



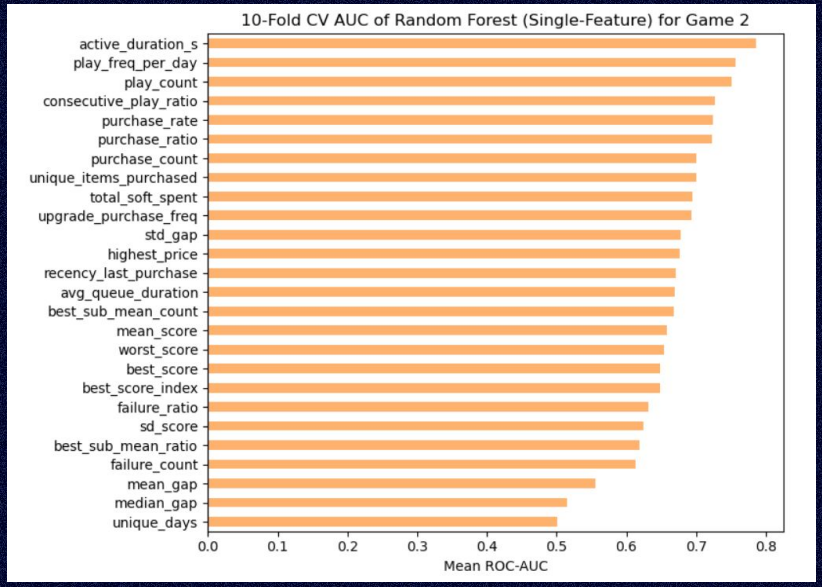


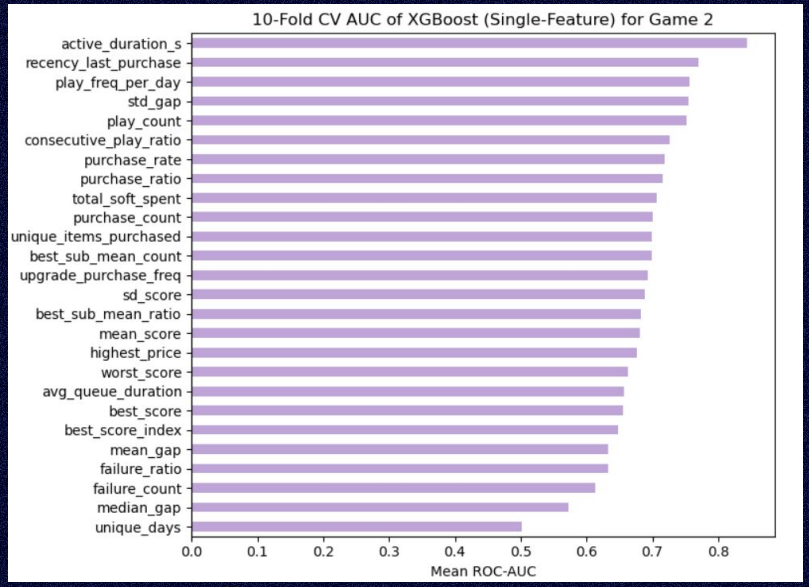
Task 6: Model Evaluation (Single Feature Ranking Charts - Game 2)





Task 6: Model Evaluation (Single Feature Ranking Charts - Game 2)





Task 6: Model Evaluation (Combined Feature Ranking - Game 1)

Combined Feature Ranking: Absolute Scores (all features)

	pearson_abs	auc_dt	auc_rf	auc_gbm	imp_gbm
play_count	0.2863	0.7639	0.7606	0.7634	0.0932
active_duration_s	0.4161	0.7742	0.5927	0.7649	0.2123
sd_gap	0.2650	0.7523	0.5818	0.7429	0.1435
consecutive_play_ratio	0.0957	0.7428	0.7391	0.7446	0.0481
avgTimeBetweenPlays	0.1735	0.7573	0.5416	0.7481	0.0459
play_frequency_per_day	0.2249	0.7375	0.7327	0.7419	0.0409
best_sub_mean_ratio	0.2656	0.7327	0.5227	0.7195	0.0425
best_score_index	0.1209	0.7061	0.6870	0.7065	0.0440
firstDayPlayCount	0.2304	0.7140	0.7090	0.7125	0.0351
scoreDeltaStd	0.1470	0.7298	0.5094	0.7169	0.0441
sd_score	0.1188	0.7177	0.5068	0.7086	0.0471
median_gap	0.1087	0.6873	0.5896	0.6800	0.0416
best_sub_mean_count	0.0497	0.6658	0.5070	0.6791	0.0424
best_score	0.1023	0.6641	0.5332	0.6457	0.0400
worst_score	0.0415	0.6119	0.5874	0.6012	0.0407
mean_score	0.0138	0.6008	0.5057	0.5756	0.0385

Combined Feature Ranking: Ranks + avg_rank (all features)

	pearson_abs_rank	auc_dt_rank	auc_rf_rank	auc_gbm_rank	imp_gbm_rank	avg_rank
play_count	2.0000	2.0000	1.0000	2.0000	3.0000	2.0000
active_duration_s	1.0000	1.0000	6.0000	1.0000	1.0000	2.0000
sd_gap	4.0000	4.0000	9.0000	5.0000	2.0000	4.8000
consecutive_play_ratio	13.0000	5.0000	2.0000	4.0000	4.0000	5.6000
avgTimeBetweenPlays	7.0000	3.0000	10.0000	3.0000	6.0000	5.8000
play_frequency_per_day	6.0000	6.0000	3.0000	6.0000	12.0000	6.6000
best_sub_mean_ratio	3.0000	7.0000	12.0000	7.0000	9.0000	7.6000
best_score_index	9.0000	11.0000	5.0000	11.0000	8.0000	8.8000
firstDayPlayCount	5.0000	10.0000	4.0000	9.0000	16.0000	8.8000
scoreDeltaStd	8.0000	8.0000	13.0000	8.0000	7.0000	8.8000
sd_score	10.0000	9.0000	15.0000	10.0000	5.0000	9.8000
median_gap	11.0000	12.0000	7.0000	12.0000	11.0000	10.6000
best_sub_mean_count	14.0000	13.0000	14.0000	13.0000	10.0000	12.8000
best_score	12.0000	14.0000	11.0000	14.0000	14.0000	13.0000
worst_score	15.0000	15.0000	8.0000	15.0000	13.0000	13.2000
mean_score	16.0000	16.0000	16.0000	16.0000	15.0000	15.8000

Task 6: Model Evaluation (Combined Feature Ranking - Game 2)

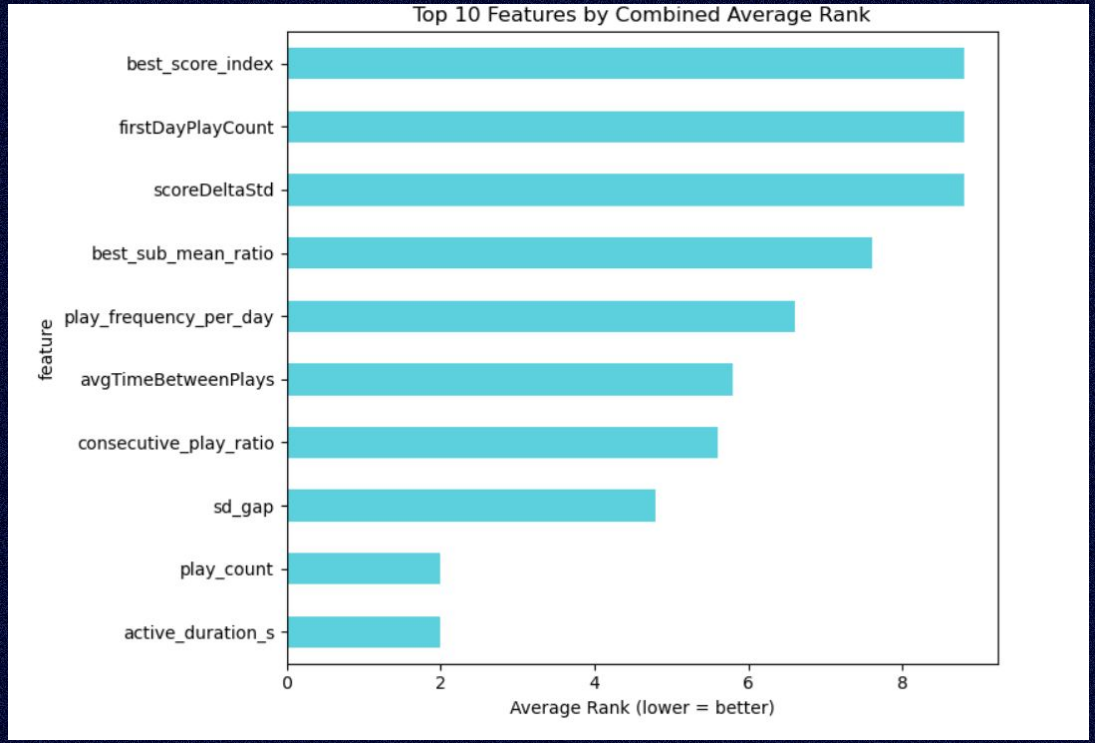
Game 2 Combined Feature Ranking: Absolute Scores (all features)

	pearson_abs	auc_dt	auc_rf	auc_gbm	imp_gbm
active_duration_s	0.5924	0.8437	0.7860	0.8440	0.5109
recency_last_purchase	0.4863	0.7703	0.6715	0.7702	0.0618
play_count	0.3385	0.7516	0.7514	0.7515	0.0215
play_freq_per_day	0.3880	0.7568	0.7569	0.7568	0.0151
std_gap	0.4204	0.7565	0.6779	0.7546	0.0175
consecutive_play_ratio	0.2403	0.7044	0.7263	0.7267	0.0168
purchase_rate	0.1039	0.6919	0.7248	0.7187	0.0197
unique_items_purchased	0.3049	0.7000	0.6997	0.6996	0.0162
total_soft_spent	0.1147	0.7034	0.6947	0.7060	0.0176
purchase_count	0.3024	0.7001	0.7001	0.7000	0.0138
upgrade_purchase_freq	0.2987	0.6935	0.6934	0.6935	0.0173
purchase_ratio	0.2150	0.6872	0.7221	0.7159	0.0161
best_sub_mean_ratio	0.1721	0.6797	0.6192	0.6820	0.0205
best_sub_mean_count	0.0026	0.6993	0.6686	0.6995	0.0189
worst_score	0.2682	0.6577	0.6545	0.6627	0.0183
highest_price	0.1472	0.6768	0.6765	0.6767	0.0172
mean_gap	0.2093	0.6332	0.5550	0.6335	0.0241
avg_queue_duration	0.0625	0.6257	0.6689	0.6564	0.0194
best_score_index	0.0029	0.6467	0.6481	0.6482	0.0216
sd_score	0.0029	0.6894	0.6247	0.6879	0.0173
failure_ratio	0.1244	0.6297	0.6316	0.6324	0.0177
failure_count	0.1741	0.6134	0.6132	0.6132	0.0189
mean_score	0.0010	0.6697	0.6579	0.6808	0.0164
median_gap	0.1296	0.5751	0.5149	0.5719	0.0205
best_score	0.0033	0.6506	0.6485	0.6548	0.0168
unique_days	0.0117	0.5009	0.5009	0.5009	0.0080

Game 2 Combined Feature Ranking: Ranks + avg_rank (all features)

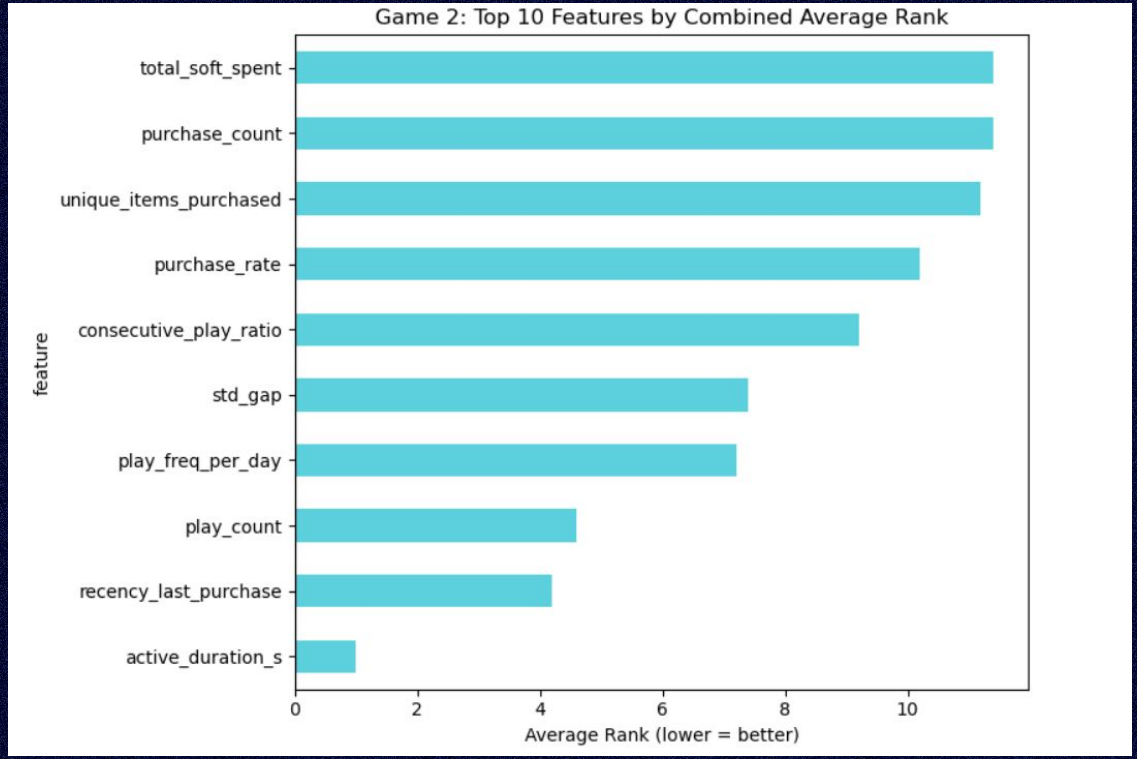
	pearson_abs_rank	auc_dt_rank	auc_rf_rank	auc_gbm_rank	imp_gbm_rank	avg_rank
active_duration_s	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
recency_last_purchase	2.0000	2.0000	13.0000	2.0000	2.0000	4.2000
play_count	5.0000	5.0000	3.0000	5.0000	5.0000	4.6000
play_freq_per_day	4.0000	3.0000	2.0000	3.0000	24.0000	7.2000
std_gap	3.0000	4.0000	11.0000	4.0000	15.0000	7.4000
consecutive_play_ratio	10.0000	6.0000	4.0000	6.0000	20.0000	9.2000
purchase_rate	19.0000	12.0000	5.0000	7.0000	8.0000	10.2000
unique_items_purchased	6.0000	9.0000	8.0000	11.0000	22.0000	11.2000
total_soft_spent	18.0000	7.0000	9.0000	9.0000	14.0000	11.4000
purchase_count	7.0000	8.0000	7.0000	10.0000	25.0000	11.4000
upgrade_purchase_freq	8.0000	11.0000	10.0000	13.0000	16.0000	11.6000
purchase_ratio	11.0000	14.0000	6.0000	8.0000	23.0000	12.4000
best_sub_mean_ratio	14.0000	15.0000	22.0000	15.0000	6.0000	14.4000
best_sub_mean_count	25.0000	10.0000	15.0000	12.0000	10.0000	14.4000
worst_score	9.0000	18.0000	17.0000	18.0000	12.0000	14.8000
highest_price	15.0000	16.0000	12.0000	17.0000	18.0000	15.6000
mean_gap	12.0000	21.0000	24.0000	22.0000	3.0000	16.4000
avg_queue_duration	20.0000	23.0000	14.0000	19.0000	9.0000	17.0000
best_score_index	24.0000	20.0000	19.0000	21.0000	4.0000	17.6000
sd_score	23.0000	13.0000	21.0000	14.0000	17.0000	17.6000
failure_ratio	17.0000	22.0000	20.0000	23.0000	13.0000	19.0000
failure_count	13.0000	24.0000	23.0000	24.0000	11.0000	19.0000
mean_score	26.0000	17.0000	16.0000	16.0000	21.0000	19.2000
median_gap	16.0000	25.0000	25.0000	25.0000	7.0000	19.6000
best_score	22.0000	19.0000	18.0000	20.0000	19.0000	19.6000
unique_days	21.0000	26.0000	26.0000	26.0000	26.0000	25.0000

Task 6: Model Evaluation (Combined Feature Ranking - Game 1)



Top 5 Features : *active_duration_s, play_count, sd_gap, consecutive_play_ratio and avgTimeBetweenPlays.*

Task 6: Model Evaluation (Combined Feature Ranking - Game 2)



Top 5 Features : *active_duration_s, recency_last_purchase, play_count, play_freq_per_day and std_gap.*

Task 6: LLM (Game 1)

Choice of LLM: Bart-Large-MNLI

Prompt: "These are the player stats from the first 5 days. You are to predict if a player will churn or not
Number of plays: ...
Total active duration (seconds): ....
Average score: ....
Score standard deviation: ...
Highest score: ...
Median time gap between plays (seconds):...
.
.
.
.
Std. deviation of score change: ... "

Task 6: LLM (Game 2)

Choice of LLM: Distilbart-MNLI-12-1

Prompt: "These are the player stats from the first 5 days. You are to predict if a player will churn or not. This is a racing game.

Number of plays: ...
Total active duration (seconds): ....
Mean Score: ....
Ratio of consecutive play days: ...
Average queue duration in seconds: ...
Purchase rate:...
.
.
.
.
Plays per day: ... "

Task 6: LLM Results Using DS2

Game 1(Bart-Large-MNLI)

Metrics	Scores
Accuracy	0.9281
Precision	0.9281
Recall	1.000
F1-Score	0.9627
ROC-AUC	0.5389

Game 2 (Distilbart-MNLI-12-1)

Metrics	Scores
Accuracy	0.4684
Precision	0.6454
Recall	0.2186
F1-Score	0.3266
ROC-AUC	0.5130

Conclusion: Scores are considerably lower than trained models since LLMS do not have game-specific context.

References

- Seungwook Kim, Daeyoung Choi, Eunjung Lee, and Wonjong Rhee. Churn prediction of mobile and online casual games using play log data. PloS one, 12(7):e0180735, 2017.

Thank You!